

Hackbuteer1的专栏

走别人没走过的路，让别人有路可走。

≡ 目录视图

≡ 摘要视图

RSS 订阅

个人资料



hackbuteer1



访问：3496909次

积分：31576

等级：

BLOG > 8

排名：第122名

原创：254篇 转载：8篇

译文：0篇 评论：2573条

文章搜索

🔍

博客专栏



考研复试上机题  
文章：14篇  
阅读：33738



IT公司笔试题集锦  
文章：18篇  
阅读：510134

文章分类

- C/C++ (63)
- OpenGL (5)
- OSG (1)
- Qt开发 (4)
- 数据结构 (61)
- OGRE (1)
- OGRE编译 (1)
- Ogitor (1)
- 编程之美 (39)
- 面试珠玑 (85)
- 九度OJ (29)

移动信息安全的漏洞和逆向原理

程序员11月书讯，评论得书啦

Get IT技能知识库，50个领域一键直达

浅谈C++多态性

标签：c++ fun 编译器 编程 语言

2012-04-18 22:45 147753人阅读 评论(75) 收藏 举报

≡ 分类： 面试珠玑 (84) ▼

| 版权声明：本文为博主原创文章，未经博主允许不得转载。

C++编程语言是一款应用广泛，支持多种程序设计的计算机编程语言。我们今天就会为大家详细介绍其中C++多态性的一些基本知识，以方便大家在学习过程中对此能够有一个充分的掌握。

多态性可以简单地概括为“一个接口，多种方法”，程序在运行时才决定调用的函数，它是面向对象编程领域的核心概念。多态(polymorphism)，字面意思多种形状。

C++多态性是通过虚函数来实现的，虚函数允许子类重新定义成员函数，而子类重新定义父类的做法称为覆盖(override)，或者称为重写。（这里我觉得要补充，重写的话可以有两种，直接重写成员函数和重写虚函数，只有重写了虚函数的才能算是体现了C++多态性）而重载则是允许有多个同名的函数，而这些函数的参数列表不同，允许参数个数不同，参数类型不同，或者两者都不同。编译器会根据这些函数的不同列表，将同名的函数的名称做修饰，从而生成一些不同名称的预处理函数，来实现同名函数调用时的重载问题。但这并没有体现多态性。

多态与非多态的实质区别就是函数地址是早绑定还是晚绑定。如果函数的调用，在编译器编译期间就可以确定函数的调用地址，并生产代码，是静态的，就是说地址是早绑定的。而如果函数调用的地址不能在编译器期间确定，需要在运行时才确定，这就属于晚绑定。

那么多态的作用是什么呢，封装可以使得代码模块化，继承可以扩展已存在的代码，他们的目的都是为了代码重用。而多态的目的则是为了接口重用。也就是说，不论传递过来的究竟是那个类的对象，函数都能够通过同一个接口调用到适应各自对象的实现方法。

最常用的用法就是声明基类的指针，利用该指针指向任意一个子类对象，调用相应的虚函数，可以根据指向的子类的不同而实现不同的方法。如果没有使用虚函数的话，即没有利用C++多态性，则利用基类指针调用相应的函数的時候，将总被限制在基类函数本身，而无法调用到子类中被重写过的函数。因为没有多态性，函数调用的地址将是一定的，而固定的地址将始终调用到同一个函数，这就无法实现一个接口，多种方法的目的了。

笔试题目：

```
[cpp]
01. #include<iostream>
02. using namespace std;
03.
04. class A
05. {
06. public:
07.     void foo()
08.     {
09.         printf("1\n");
10.     }
11.     virtual void fun()
12.     {
13.         printf("2\n");
14.     }
15. };
16. class B : public A
17. {
18. public:
```

STL源码剖析 (9)

剑指Offer (21)

Spring笔记 (0)

文章存档

2014年09月 (1)

2013年09月 (9)

2013年06月 (1)

2013年05月 (2)

2013年01月 (4)

展开

阅读排行

浅谈C++多态性 (147687)

C++中的单例模式 (123147)

百度最新面试题集锦 (116277)

C++中智能指针的设计和链表各类操作详解 (82475)

字符串的全排列和组合 (72892)

程序员有趣的面试智力题 (68900)

虚函数和纯虚函数的区别 (68694)

N皇后问题的两个最高效 (63141)

阿里巴巴笔试题 (60272)

阿里巴巴笔试题 (56446)

评论排行

程序员有趣的面试智力题 (153)

淘宝2011.9.21校园招聘 (118)

百度最新面试题集锦 (113)

阿里巴巴笔试题 (112)

浅谈C++多态性 (75)

C++中智能指针的设计和C++中的单例模式 (66)

一个应届计算机毕业生的 (59)

网易游戏2011.10.15校园 (53)

微软校园招聘笔试题 (51)

推荐文章

\* 程序员10月书讯、评论得书

\* Android中Xposed框架篇---修改系统位置信息实现自身隐藏功能

\* Chromium插件（Plugin）模块（Module）加载过程分析

\* Android TV开发总结--构建一个TV app的直播节目实例

\* 架构设计：系统存储--MySQL简单主从方案及暴露的问题

最新评论

浅谈C++多态性

离光曦: 感谢博主分享, 但感觉有个地方说的不是很准确, 在派生类中写和基类函数名相同而参数列表不同的函数, 不是函...

C++中虚函数工作原理和(虚)继承 wyx\_123456: 楼主, 请问虚函数

```
19.     void foo()
20.     {
21.         printf("3\n");
22.     }
23.     void fun()
24.     {
25.         printf("4\n");
26.     }
27. };
28. int main(void)
29. {
30.     A a;
31.     B b;
32.     A *p = &a;
33.     p->foo();
34.     p->fun();
35.     p = &b;
36.     p->foo();
37.     p->fun();
38.     return 0;
39. }
```

第一个p->foo()和p->fun()都很好理解, 本身是基类指针, 指向的又是基类对象, 调用的都是基类本身的函数, 因此2。

第二个输出结果就是1、4。p->foo()和p->fun()则是基类指针指向子类对象, 正式体现多态的用法, p->foo()由于指针是个基类指针, 指向是一个固定偏移量的函数, 因此此时指向的就只能是基类的foo()函数的代码了, 因此输出的结果还是1。而p->fun()指针是基类指针, 指向的fun是一个虚函数, 由于每个虚函数都有一个虚函数列表, 此时p调用fun()并不是直接调用函数, 而是通过虚函数列表找到相应的函数的地址, 因此根据指向的对象不同, 函数地址也将不同, 这里将找到对应的子类的fun()函数的地址, 因此输出的结果也会是子类的结果4。

笔试的题目中还有一个另类测试方法。即

**B \*ptr = (B \*)&a; ptr->foo(); ptr->fun();**

问这两调用的输出结果。这是一个用子类的指针去指向一个强制转换为子类地址的基类对象。结果, 这两句调用的输出结果是3, 2。

并不是很理解这种用法, 从原理上来解释, 由于B是子类指针, 虽然被赋予了基类对象地址, 但是ptr->foo()在调用的时候, 由于地址偏移量固定, 偏移量是子类对象的偏移量, 于是即使在指向了一个基类对象的情况下, 还是调用到了子类的函数, 虽然可能从始至终都没有子类对象的实例化出现。

而ptr->fun()的调用, 可能还是因为C++多态性的原因, 由于指向的是一个基类对象, 通过虚函数列表的引用, 找到了基类中fun()函数的地址, 因此调用了基类的函数。由此可见多态性的强大, 可以适应各种变化, 不论指针是基类的还是子类的, 都能找到正确的实现方法。

```
[cpp] 复制
01. //小结：1、有virtual才可能发生多态现象
02. // 2、不发生多态（无virtual）调用就按原类型调用
03. #include<iostream>
04. using namespace std;
05.
06. class Base
07. {
08. public:
09.     virtual void f(float x)
10.     {
11.         cout<<"Base::f(float)"<< x <<endl;
12.     }
13.     void g(float x)
14.     {
15.         cout<<"Base::g(float)"<< x <<endl;
16.     }
17.     void h(float x)
18.     {
19.         cout<<"Base::h(float)"<< x <<endl;
20.     }
21. };
22. class Derived : public Base
23. {
24. public:
25.     virtual void f(float x)
26.     {
27.         cout<<"Derived::f(float)"<< x <<endl; //多态、覆盖
28.     }
29.     void g(int x)
```

表指针算不算对象的数据成员？

STL源码剖析---vector

lingen1949: 请问下insert\_aux里面的if (finish != end\_of\_stora...

链表各类操作详解

xyw\_1122: 楼主，能否发个原稿给我，最近学链表觉得老是弄不清楚，看你写的好详细~麻烦楼主了~1228627366...

C++中的static关键字

kongxian2007: 总结很全面，谢谢啦

程序员面试100题之七：最长公共

HelloBob: c表示Xi和Yi的最大Substring的长度,是不是说法有些问题.X4和 Y4的最大长度是2啊

C++中智能指针的设计和使用的

little\_face: void set\_ptr(int \*p) const { ptr->ip...

C++中的单例模式

break:: @Shirley:学习一下。我只知道进程结束了，系统会释放内存。你这里说他不会调用delete的意...

一个应届计算机毕业生的2012求

Sin是我是我: DS躺枪

分治算法求最近点对

zhuan\_2: 不明白  
printf("%0.2f\n",closest(0, n - 1)/2);为什么除以2, ...

```
30.     {
31.         cout<<"Derived::g(int)"<< x <<endl;    //隐藏
32.     }
33.     void h(float x)
34.     {
35.         cout<<"Derived::h(float)"<< x <<endl;    //隐藏
36.     }
37. };
38. int main(void)
39. {
40.     Derived d;
41.     Base *pb = &d;
42.     Derived *pd = &d;
43.     // Good : behavior depends solely on type of the object
44.     pb->f(3.14f);    // Derived::f(float) 3.14
45.     pd->f(3.14f);    // Derived::f(float) 3.14
46.
47.     // Bad : behavior depends on type of the pointer
48.     pb->g(3.14f);    // Base::g(float) 3.14
49.     pd->g(3.14f);    // Derived::g(int) 3
50.
51.     // Bad : behavior depends on type of the pointer
52.     pb->h(3.14f);    // Base::h(float) 3.14
53.     pd->h(3.14f);    // Derived::h(float) 3.14
54.     return 0;
55. }
```

令人迷惑的隐藏规则

本来仅仅区别重载与覆盖并不算困难，但是C++的隐藏规则使问题复杂性陡然增加。

这里“隐藏”是指派生类的函数屏蔽了与其同名的基类函数，规则如下：

- (1) 如果派生类的函数与基类的函数同名，但是参数不同。此时，不论有无virtual关键字，基类的函数将被隐藏（注意别与重载混淆）。
  - (2) 如果派生类的函数与基类的函数同名，并且参数也相同，但是基类函数没有virtual关键字。此时，基类的函数被隐藏（注意别与覆盖混淆）。
- 上面的程序中：
- (1) 函数Derived::f(float)覆盖了Base::f(float)。
  - (2) 函数Derived::g(int)隐藏了Base::g(float)，而不是重载。
  - (3) 函数Derived::h(float)隐藏了Base::h(float)，而不是覆盖。

## C++纯虚函数

### 一、定义

纯虚函数是在基类中声明的虚函数，它在基类中没有定义，但要求任何派生类都要定义自己的实现方法。在基类中实现纯虚函数的方法是在函数原型后加“=0”

```
virtual void function()=0
```

### 二、引入原因

- 1、为了方便使用多态特性，我们常常需要在基类中定义虚拟函数。
- 2、在很多情况下，基类本身生成对象是不合理的。例如，动物作为一个基类可以派生出老虎、孔雀等子类，但动物本身生成对象明显不合理。

为了解决上述问题，引入了纯虚函数的概念，将函数定义为纯虚函数（方法：virtual Return Type Function()= 0;），则编译器要求在派生类中必须予以重写以实现多态性。同时含有纯虚拟函数的类称为抽象类，它不能生成对象。这样就很好地解决了上述两个问题。

### 三、相似概念

#### 1、多态性

指相同对象收到不同消息或不同对象收到相同消息时产生不同的实现动作。C++支持两种多态性：编译时多态性，运行时多态性。

**a、编译时多态性：通过重载函数实现**

**b、运行时多态性：通过虚函数实现。**

#### 2、虚函数

虚函数是在基类中被声明为virtual，并在派生类中重新定义的成员函数，可实现成员函数的动态覆盖（Override）

#### 3、抽象类

包含纯虚函数的类称为抽象类。由于抽象类包含了没有定义的纯虚函数，所以不能定义抽象类的对象。

顶

踩

163

1

上一篇

C++经典面试题

下一篇

N\*N匹马，N个赛道，求出最快N匹马的解法

我的同类文章

面试珠玑 (84)					
• 2015届华为校园招聘机试题	2014-09-13	阅读 44561	• 迅雷2014校园招聘笔试题	2013-09-09	阅读 44129
• PPS2013校园招聘笔试题	2013-09-09	阅读 7479	• 2013豆瓣校园招聘研发类笔...	2013-09-06	阅读 44129
• 网新恒天2013年校园招聘笔试	2013-09-06	阅读 8085	• 2013届华为校园招聘机试题	2013-09-05	阅读 9327
• 2014届华为校园招聘机试题	2013-09-05	阅读 39768	• 各大IT公司校园招聘程序猿...	2012-09-09	阅读 30757
• C/C++笔试题目大全	2012-08-30	阅读 35214	• C++中虚函数工作原理和(虚)...	2012-08-19	阅读 43457
• 大端模式和小端模式	2012-07-06	阅读 39017			

猜你在找

C++语言基础	浅谈C++多态性
汇编C语言C++基础原理系列经典教程	浅谈C++多态性
c++面向对象前言及意见征集（来者不拒）视频课程	浅谈C++多态性
《C语言/C++学习指南》语法篇（从入门到精通）	浅谈C++多态性
Visual Studio 2015开发C++程序的基本使用	浅谈C++多态性

查看评论

- 57楼 离光曦 3天前 18:39发表
-  感谢博主分享，但感觉有个地方说的不是很准确，在派生类中写和基类函数名相同而参数列表不同的函数，不是函数重载，而是函数隐藏，二者还是有区别的，重载是在同一个名字空间作用域里定义的，隐藏则不是。
- 56楼 caisam 2016-09-21 09:52发表
-  多谢分享
- 55楼 xfyae 2016-08-30 19:05发表
-  多谢楼主分享。
- 54楼 勤奋的Garfield 2016-08-28 01:25发表
-  B b; //b的内存空间存储情况为【base class成员变量】+【derived class成员变量】+【指向虚表的指针】， 本题省略成员变量，因此b中仅仅包含【指向虚表的指针】， 并且其显然仅仅指向重载后的函数。  
A\* a=&b; //这种指向不会修改任何b对象的内容  
a->foo();输出是1， 是因为b对象内容并不包含任何关于普通成员函数foo()的信息， 究竟调用父类还是子类的函数完全取决于对象指针的类型；a->fun()输出是4， 是因为fun()是个虚函数， 在调用时， 需要根据“指向虚表的指针”， 来找到虚函数， b的虚表“指向虚表的指针”指向的函数是重载后的函数， 因为会输出4。  
  
-----  
  
A a;  
B\* b=(B\*) &a; // 不修改任何a对象的内容  
b->foo(); 输出3， 这种普通成员函数调用子类还是父类的函数完全取决于指针的类型， 因为a中不包含任何关于foo()的信息。  
  
  
b->fun(); 输出2， 。。。根据“指向虚表的指针”来调用虚函数， a中虚表指向的是a的虚函数， 所以输出是2

<http://www.zhihu.com/question/25572937>

Re: 勤奋的Garfield 2016-08-28 01:28发表



回复zhang\_jinhe：第一种情况的a->fun();的输出应该说完全取决于什么类型的对象调用fun(); 请注意"a->fun()"中的a

53楼 [xiangjai](#) 2016-08-22 11:35发表



B \*ptr = (B \*)&a; ptr->foo(); ptr->fun();

代码可以从内存分布上理解

类A和B 中， 函数的内存空间是公用的， 变量都是私有的空间， 指针调用的函数时间上就是A函数空间上的

52楼 [playboy\\_lei](#) 2016-07-18 19:14发表



写的太好了，受教！

51楼 [\\_Kite](#) 2016-07-17 23:05发表



谢谢。受益匪浅。

50楼 [time770880](#) 2016-07-13 09:10发表



感谢分享 收获很大

49楼 [xiang90721](#) 2016-06-01 15:11发表



B \*ptr = (B \*)&a; ptr->foo(); ptr->fun();

这部分还有其他解释么，不是很明白

48楼 [lishuandao](#) 2016-03-20 20:42发表



谢谢你，写的非常清楚，以后有问题再向你请教。

47楼 [peerless0909](#) 2015-12-05 00:01发表



易懂

46楼 [怪叔叔萝莉控](#) 2015-11-30 22:00发表



写的特别好，谢谢啦，转走～

45楼 [heyang408856755](#) 2015-08-04 11:02发表



我用VC6.0和VS2005都是3.2

44楼 [This is bill](#) 2015-07-25 17:00发表



(3) 函数Derived::h(float)隐藏了Base::h(float)，而不是覆盖。为什么

43楼 [markjenny](#) 2015-07-21 20:41发表



关于名字覆盖那里，也就是楼主说的“隐藏”，在《effective C++》中有写：无论函数时什么样子的，只要是基类和派生类中的对象相同，派生类中的就会对基类中同名的函数进行覆盖。（一看已经是2012年的博文了。。。不过还是说一下我自己的观点，哈哈

还有“ B \*ptr = (B \*)&a; ptr->foo(); ptr->fun();”，我认为这个之所以使“3.2”是因为对于foo， ptr是静态判定，也就是编译器确定的东西，那么他是调用派生类B中的foo()是没有问题的。

42楼 - 破天一剑 - 2015-06-04 10:28发表



简单的例子：

子类们只想重写某个函数，用各自的对象访问这个函数。这时，不用多态可以，用也行

想让父类的指针或引用指向不同子类的对象，还想调用某个函数，还要由不同的“特性”，这时，使用多态

那么使用多态总不会出错，为什么C++不默认使用多态呢？

难道是因为运行时效率影响太大？

再者，其实是否想得到想要的输出，多态的意图完全可以在编译的时候确定，为什么要等到运行时。比如：

```
class A
{
public:
```

```
virtual void print() {
    cout << "This is A\n";
}
};
class B
{
public:
};
class C
{
public:
};

int main() {
    C c;
    A * a = &c;
    a->print();
}
```

这里"极端的例子"中，使用了虚函数实现多态。编译器在编译时完全可以确定，a->print()调用的那个方法其实是在类A里面。（它能发现c是C类的对象，嗯，到类C的声明中，没找到 print？好，递归到父类B中找，还是没有print？好，再递归，找到了A类，A类中有print。行了，那编译器不就确定了这个方法的位置了么？不就不用动态确定了么？）

如果C++不默认使用多态是害怕运行时消耗太多时间的话，那么为什么不让编译器在编译的时候确定？既然编译器可以确定。那么C++为什么不默认使用多态？  
如果是这样，就没有那么多知识的问题了。

如果有错误，请博主指点，我只是刚学习的新手。

如果没有错误，这是C++的缺陷吗？

41楼 - [\\_破天一剑\\_](#) - 2015-06-04 10:28发表



简单的例子：

子类们只想重写某个函数，用各自的对象访问这个函数。这时，不用多态可以，用也行

想让父类的指针或引用指向不同子类的对象，还想调用某个函数，还要由不同的"特性"，这时，使用多态

那么使用多态总不会出错，为什么C++不默认使用多态呢？  
难道是因为运行时效率影响太大？

再者，其实是否想得到想要的输出，多态的意图完全可以在编译的时候确定，为什么要等到运行时。比如：

```
class A
{
public:
    virtual void print() {
        cout << "This is A\n";
    }
};
class B
{
public:
};
class C
{
public:
};

int main() {
    C c;
    A * a = &c;
    a->print();
}
```

这里"极端的例子"中，使用了虚函数实现多态。编译器在编译时完全可以确定，a->print()调用的那个方法其实是在类A里面。（它能发现c是C类的对象，嗯，到类C的声明中，没找到 print？好，递归到父类B中找，还是没有print？好，再递归，找到了A类，A类中有print。行了，那编译器不就确定了这个方法的位置了么？不就不用动态确定了么？）

如果C++不默认使用多态是害怕运行时消耗太多时间的话，那么为什么不让编译器在编译的时候确定？既然编译器可以确定。那么C++为什么不默认使用多态？  
如果是这样，就没有那么多知识的问题了。

如果有错误，请博主指点，我只是刚学习的新手。

如果没有错误，这是C++的缺陷吗？

40楼 - [\\_破天一剑\\_](#) - 2015-06-04 10:27发表



简单的例子：

子类们只想重写某个函数，用各自的对象访问这个函数。这时，不用多态可以，用也行

想让父类的指针或引用指向不同子类的对象，还想调用某个函数，还要由不同的“特性”，这时，使用多态

那么使用多态总不会出错，为什么C++不默认使用多态呢？

难道是因为运行时效率影响太大？

再者，其实是否想得到想要的输出，多态的意图完全可以在编译的时候确定，为什么要等到运行时。比如：

```
class A
{
public:
virtual void print() {
cout << "This is A\n";
}
};
class B
{
public:
};
class C
{
public:
};

int main() {
C c;
A * a = &c;
a->print();
}
```

这里“极端的例子”中，使用了虚函数实现多态。编译器在编译时完全可以确定，a->print()调用的那个方法其实是在类A里面。

（它能发现c是C类的对象，嗯，到类C的声明中，没找到 print？好，递归到父类B中找，还是没有print？好，再递归，找到了A类，A类中有print。行了，那编译器不就确定了这个方法的位置了么？就不用动态确定了么？）

如果C++不默认使用多态是害怕运行时消耗太多时间的话，那么为什么不让编译器在编译的时候确定？既然编译器可以确定。

那么C++为什么不默认使用多态？

如果是这样，就没有那么多知识的问题了。

如果有错误，请博主指点，我只是刚学习的新手。

如果没有错误，这是C++的缺陷吗？

39楼 - [\\_破天一剑\\_](#) - 2015-06-04 10:26发表



简单的例子：

子类们只想重写某个函数，用各自的对象访问这个函数。这时，不用多态可以，用也行

想让父类的指针或引用指向不同子类的对象，还想调用某个函数，还要由不同的“特性”，这时，使用多态

那么使用多态总不会出错，为什么C++不默认使用多态呢？

难道是因为运行时效率影响太大？

再者，其实是否想得到想要的输出，多态的意图完全可以在编译的时候确定，为什么要等到运行时。比如：

```
class A
{
public:
virtual void print() {
cout << "This is A\n";
}
};
class B
{
public:
};
class C
{
```

```
public:
};
```

```
int main() {
    C c;
    A * a = &c;
    a->print();
}
```

这里"极端的例子"中, 使用了虚函数实现多态。编译器在编译时完全可以确定, `a->print()`调用的那个方法其实是在类A里面。  
(它能发现c是C类的对象, 嗯, 到类C的声明中, 没找到 `print` ? 好, 递归到父类B中找, 还是没有`print` ? 好, 再递归, 找到了A类, A类中有`print`。行了, 那编译器不就确定了这个方法的位置了么? 就不用动态确定了么?)

如果C++不默认使用多态是害怕运行时消耗太多时间的话, 那么为什么不让编译器在编译的时候确定? 既然编译器可以确定。  
那么C++为什么不默认使用多态?  
如果是这样, 就没有那么多知识的问题了。

如果有错误, 请博主指点, 我只是刚学习的新手。

如果没有错误, 这是C++的缺陷吗?

Re: [sandyznb](#) 2015-06-25 19:09发表



回复zhangpzh5: 你这你编译过了么?  
你还是好好看看c++吧

38楼 [qinpengtaiyuan](#) 2015-03-30 15:07发表



好文章! 不过隐藏规则部分有误。第一条隐藏规则存在 (<http://isocpp.org/wiki/faq/strange-inheritance#hiding-rule>) ; 第二条隐藏规则不存在 (<http://isocpp.org/wiki/faq/strange-inheritance#redefining-nonvirtuals>) , 属于对非虚函数的重定义。

37楼 [liuyu1254](#) 2015-01-23 19:19发表



我使用vs2012运行时, 结果是3,4  
不是楼主说的3,2

```
#include<iostream>
using namespace std;
```

```
class A
{
public:
    void foo()
    {
        printf("1\n");
    }
    virtual void fun()
    {
        printf("2\n");
    }
};
class B : public A
{
public:
    void foo()
    {
        printf("3\n");
    }
    void fun()
    {
        printf("4\n");
    }
};
```

```
int main(void)
{
    A a;
    B b;
```

```
    /*A *p = &a;
    p->foo();
    p->fun();
    p = &b;
    p->foo();
```



```
p->fun();*/
```

```
B *ptr = (B *)&a;  
ptr->foo();  
ptr->fun();
```

```
return 0;  
}
```

我使用vs2012运行时，结果是3,4  
不是楼主说的3,2  
希望楼主及时改正，整篇文章写得还是蛮好的

Re: [中古遗人](#) 2015-04-21 17:44发表



回复 liuyu1254：结果是 3 2

Re: [perthblank](#) 2015-03-12 15:23发表



回复 liuyu1254：VS2012 也是 32

Re: [北辰剑心](#) 2015-02-26 16:33发表



回复 liuyu1254：我用GCC编译器也是3，2 难道VS2012的编译器做了改动？

Re: [liumingd](#) 2015-01-26 10:15发表



回复 liuyu1254：我在GCC 4.8.2和VC++ express 2010上运行，结果都是3，2

36楼 [yanghan199110](#) 2015-01-05 23:24发表



polymorphism  
楼主写成polymorphisn

35楼 [无话可说RD](#) 2014-12-09 15:33发表



谢谢分享。。。。。

34楼 [渲染大师](#) 2014-11-08 19:31发表



大神，看了你的几篇博客，好牛逼啊，准备向你学习，虽然晚了点，但是相信只要努力了就会有收获，路还很长。

33楼 [qq\\_18753877](#) 2014-08-11 17:50发表



非常感谢，加深了我对多态性的认识

32楼 [clqwt](#) 2014-07-12 18:36发表



讲的不错

31楼 [SmartSmall](#) 2014-05-22 00:27发表



mark

30楼 [wdlove58](#) 2014-05-07 20:58发表



很好，学习了，总算对多态有了清晰的认识了

29楼 [taipingsu](#) 2014-04-29 09:50发表



分析得很好，又学习受教了

28楼 [码亮虔诚](#) 2014-03-23 10:19发表



分析和考虑的很全面，写的也很容易理解，赞

27楼 [蒲公英小帝](#) 2014-03-21 15:19发表



学习了

26楼 [trista\\_ning](#) 2014-03-13 18:02发表



帮助挺大的，谢谢了

25楼 [云荒侠客](#) 2014-02-24 14:07发表



厉害

24楼 [阿奴波仔](#) 2014-02-05 09:21发表



B \*ptr = (B \*)&a; ptr->foo(); ptr->fun();

对于这个调用结果，我是这样理解的：

成员函数和类本身并不是完全绑定的，在对象调用成员函数的时候，还会隐式的把自己的 this 指针传过去，也就是说，函数的参数里面有 this 指针的类型。这样基类和子类的同名函数就可以用重载来理解了。

在 ptr->foo(); 的时候，虽然 ptr 是用父类的对象地址初始化，但本身还是子类的指针，所以调用的时候编译器会找到子类的 foo() 函数

请问前辈这个理解有没有问题

23楼 [法哥的铲铲队](#) 2013-12-23 10:39发表



非常好，转走了

22楼 [苍原狮啸](#) 2013-11-30 10:53发表



继承是实现代码的可重用性，这话很好理解。可以直接调用父类的函数。多态是实现接口的可重用性，这话就不那么好懂了。多态仅仅实现了接口的标准化，规定了继承类必须按规则行事，但同时也把父类给抽象化了（纯虚函数），对于纯虚函数其本身并没有实现继承功能啊，（不包含任何代码，还把父类的实例化给牺牲了），相比于覆盖或者隐藏的特性，对纯虚函数实现多态的理解并不是那么好懂

21楼 [gordon1986](#) 2013-10-14 18:00发表



mark一下，待会来详细读

20楼 [天行者pxhero](#) 2013-10-04 10:37发表



多态分为静态编译时多态和运行时多态，函数重载算静态编译时多态！此话出自C++ PROGRAMMING！C++创始人说的。

19楼 [没见过猪跑](#) 2013-09-15 10:19发表



好啊，解惑了，谢谢。

18楼 [Rain-晴天](#) 2013-08-18 09:32发表



好文章啊，支持一下，之前老是弄不清他的概念

17楼 [lz2013](#) 2013-07-12 16:33发表



理解了多态概念了，很好！顶！

16楼 [逍遥恣](#) 2013-06-05 22:56发表



收益良多,感谢分享...

15楼 [fuyong13](#) 2013-05-28 08:41发表



受教

14楼 [云斜月](#) 2013-05-23 22:36发表



关于 B \*ptr = (B \*)&a; ptr->foo(); ptr->fun(); 主要涉及到两类多态的问题，一个是编译时多态，一个是运行时多态，non-virtual 函数是执行的编译时多态，virtual是运行时多态。

对non-virtual成员函数的调用由指针/引用的类型决定。

而virtual 成员函数则有赋值类型决定。

13楼 [mjay1234](#) 2013-04-22 10:24发表



弱弱的问一句：覆盖和隐藏有什么不一样，请楼主不吝赐教~

Re: [jhq1204](#) 2013-08-10 13:50发表



回复mjay1234：覆盖需要满足的条件：

1. 基类中的函数要为virtual函数；

2. 子类中的同名函数要和父类中的这个virtual函数有相同的函数原型（包括返回值）；这样才能构成覆盖（或叫重写）。所以我觉得这个概念只会出现在多态中。

而对于隐藏，只是说子类中存在和父类中同名的方法，不管其形参列表和返回值是否一样，都导致父类中的同名方法被屏蔽。

12楼 湖边的小牛 2013-04-18 18:24发表



指针类型对这个 指针的指向有什么影响？？基类指针指向子类还能理解、子类指针指向父类就不理解了。

11楼 邱天 2013-04-18 09:39发表



关于继承的方法是否会重载，推荐楼主看一下这篇文章：  
<http://blog.csdn.net/lenotang/article/details/2681525>

Re: 梦想的IT男 2013-10-10 20:11发表



回复RQSLT：重载是在一个类中出现；重写是在多个类中。  
继承出现在多个类中，不会出现重载现象。

10楼 InfoStation信息站 2013-04-10 19:08发表



"只有重写了虚函数的才能算是体现了C++多态性"？  
c++的多态分为运行时多态和编译时多态，编译时多态是通过普通函数重载实现包括运算符重载以及模板体现的；而运行时多态则是通过虚函数体现的。

Re: sandyzn 2015-06-25 17:34发表



回复zhangshuaizaxia：楼主的意思是体现运行时多态，不必咬文嚼字

9楼 闭着眼刷牙 2013-01-28 17:01发表



帮我理解多态的概念 谢谢

8楼 奇异果 2012-11-22 20:02发表



在流行的C++教科书中，对多态有两种截然不同的说法，一种认为：“因为多态性增加了一些数据存储和执行指令的代价，所以能不多态最好。”另一种认为：“这是类机制中最关键的一个方面。只要用的好，它能够成为面向对象程序设计的基石”（引号中的文字均引自原文）你赞同哪种说法

7楼 a304672343 2012-11-08 09:27发表



学习了！  
编译时多态性：通过重载函数实现///除了函数，还包括运算符重载。

6楼 Pluser 2012-09-27 16:10发表



前辈你这里说 多态与非多态的本质区别就是  
看其是动态编译还是静态编译，而多态是动态编译。

那么又说：C++多态性分静态多态即重载和模板，动态多态即覆盖。

这两句话不冲突么？其实我一直认为第2种正确，但是最近又看到一些书说第一种。所以想问下前辈。  
希望能尽快回答，再次非常感谢

5楼 abc609315767 2012-09-17 11:23发表



既然说子类指针指向子类对象，那我通过子类指针想调用父类中继承而来的非virtual同名函数。想要突破隐藏规则，应该如何做到？请楼主给个解释。

Re: hackbuteer1 2012-09-19 11:13发表



回复abc609315767：要突破隐藏规则的话，可以不要定义同名的函数，否则就会被隐藏的

Re: abc609315767 2012-09-23 11:48发表




回复Hackbuteer1：无意中在深入浅出MFC书中看到突破方法，其实可以实现！例：A为基类 B为子类，A中有非visual show()方法，B也定义也同名函数，把A中的show()隐藏了，我们还是可以直接调用隐藏的show()方法的。  
B b;  
b.A::show(); //这样就直接突破了相应的隐藏规则的限制了

Re: sandyzn 2015-06-25 17:30发表




回复abc609315767：这方法挺好，第一次见


4楼 Zenseven 2012-09-13 11:09发表

 前面写的很好，到后面就出错了。  
关于重载、重写（覆盖）、隐藏 请参考  
<http://www.cnblogs.com/qlee/archive/2011/07/04/2097055.html>


Re: hackbuteer1 2012-09-13 14:40发表

 回复hbissinper：是的，弄错了，修改过来了

Re: Zenseven 2012-09-14 07:01发表

 回复Hackbuteer1：呵呵，相信楼主，我更加钦佩你了！

3楼 Zenseven 2012-09-13 11:07发表




```


1.  public:
2.      virtual void f(float x)
3.      {
4.          cout<<"Derived::f(float)"<< x <<endl;    //多态(覆盖)必须不同类
5.      }
6.      void g(int x)
7.      {
8.          cout<<"Derived::g(int)"<< x <<endl;      //隐藏（参数不同。此时，不论有无virtual关键字，基类的函数将被隐藏（注意别与重载混淆）。）
9.      }
10.     void h(float x)
11.     {
12.         cout<<"Derived::h(float)"<< x <<endl;    //隐藏（参数也相同，但是基类函数没有virtual关键字。
13.     }
                                     //此时，基类的函数被隐藏（注意别与覆盖混淆）。）

```

2楼 zhang\_g\_y 2012-09-12 21:24发表


 对我理解多态和虚函数帮助很大啊，非常感谢。

1楼 qiuyang0607 2012-04-20 09:58发表

 "2、虚函数  
虚函数是在基类中被声明为virtual，并在派生类中重新定义的成员函数，可实现成员函数的动态重载"

这个“动态重载”是不是改为“动态覆盖或动态重写”比较好一些。

Re: hackbuteer1 2012-04-20 11:12发表

 回复qiuyang0607：的确啊，是的，重载的说法不太好，应该是Override（覆盖）。。

您还没有登录,请[登录](#)或[注册](#)

\* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题   Hadoop   AWS   移动游戏   Java   Android   iOS   Swift   智能硬件   Docker   OpenStack  
VPN   Spark   ERP   IE10   Eclipse   CRM   JavaScript   数据库   Ubuntu   NFC   WAP   jQuery  
BI   HTML5   Spring   Apache   .NET   API   HTML   SDK   IIS   Fedora   XML   LBS   Unity  
Splashtop   UML   components   Windows Mobile   Rails   QEMU   KDE   Cassandra   CloudStack   FTC  
coremail   OPhone   CouchBase   云计算   iOS6   Rackspace   Web App   SpringSide   Maemo  
Compuware   大数据   aptech   Perl   Tornado   Ruby   Hibernate   ThinkPHP   HBase   Pure   Solr  
Angular   Cloud Foundry   Redis   Scala   Django   Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服   杂志客服   微博客服   webmaster@csdn.net   400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 |

江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2016, CSDN.NET, All Rights Reserved

